Módulo II

Comunicaciones seguras

- Tema 3: Protección de la confidencialidad
- Tema 4: Cifrados de clave secreta
- Tema 5: Distribución de claves

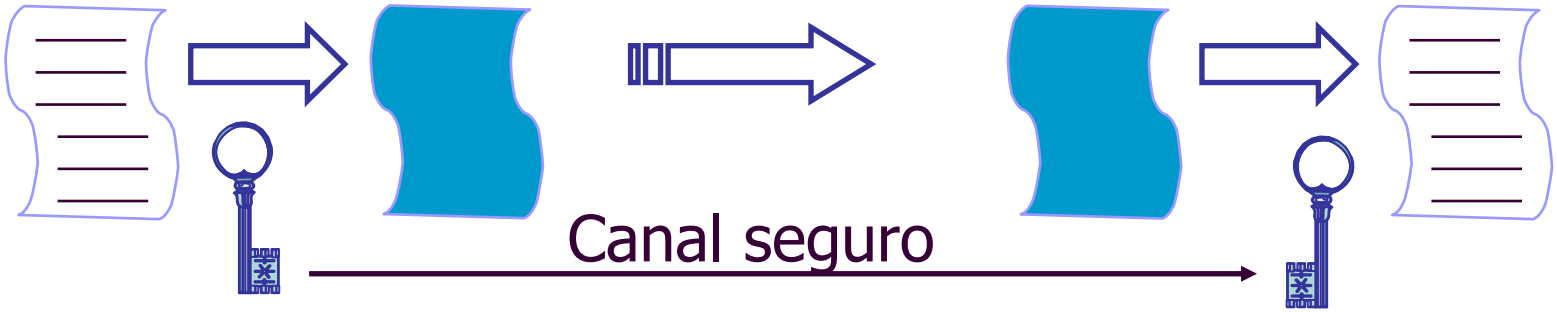
Comunicaciones Seguras

En:

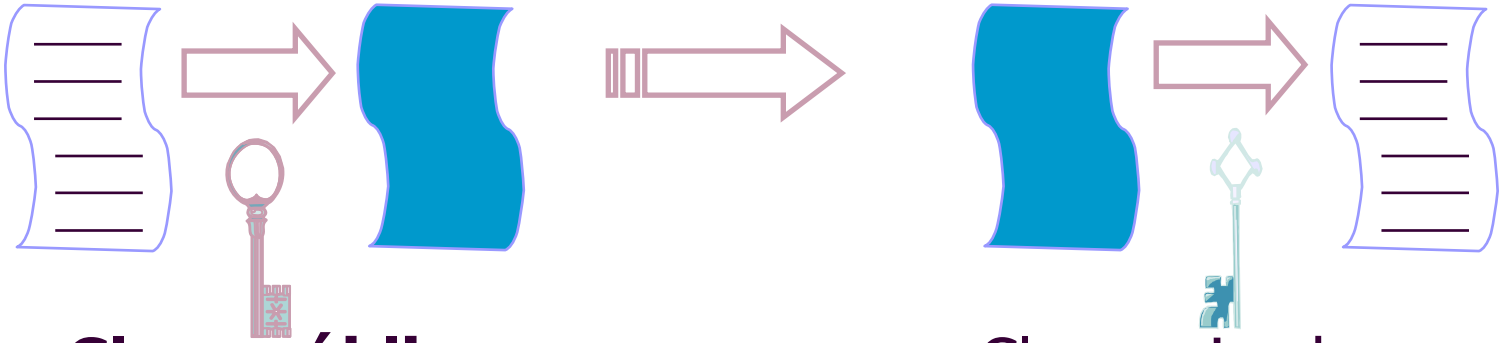
- Wi-Fi
- SSL/TLS
- GPS
- BLE y NFC
- Telefonía Móvil

Criptografía Moderna

– **Criptosistemas Simétricos o de Clave Secreta:**

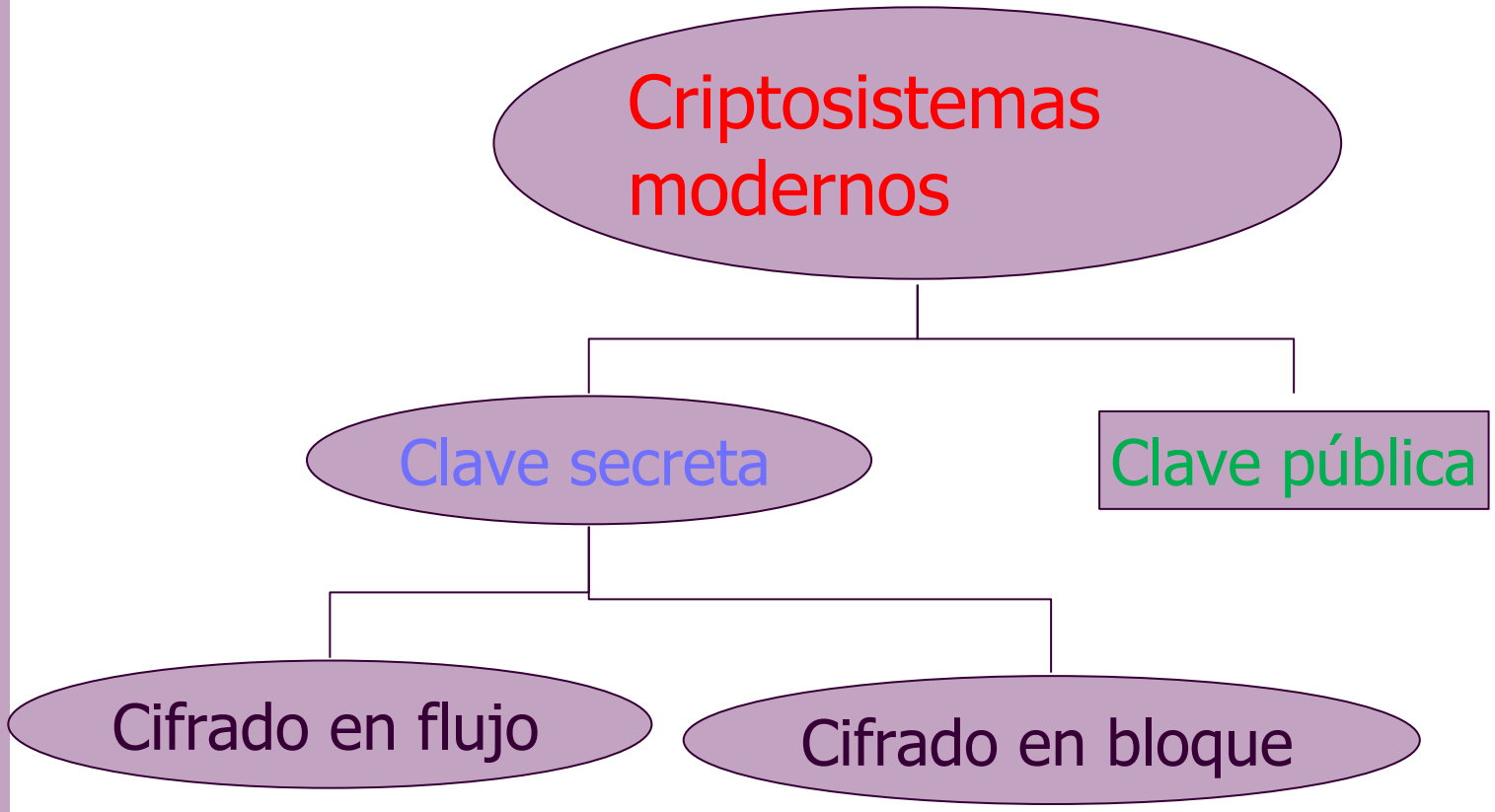


– **Criptosistemas Asimétricos o de Clave Pública:**

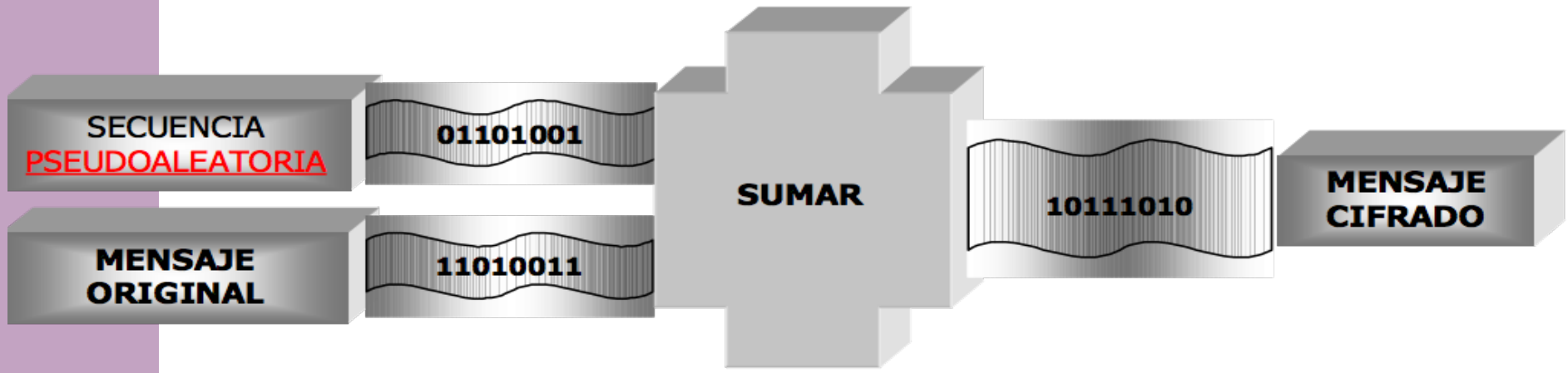


Clave pública Clave privada

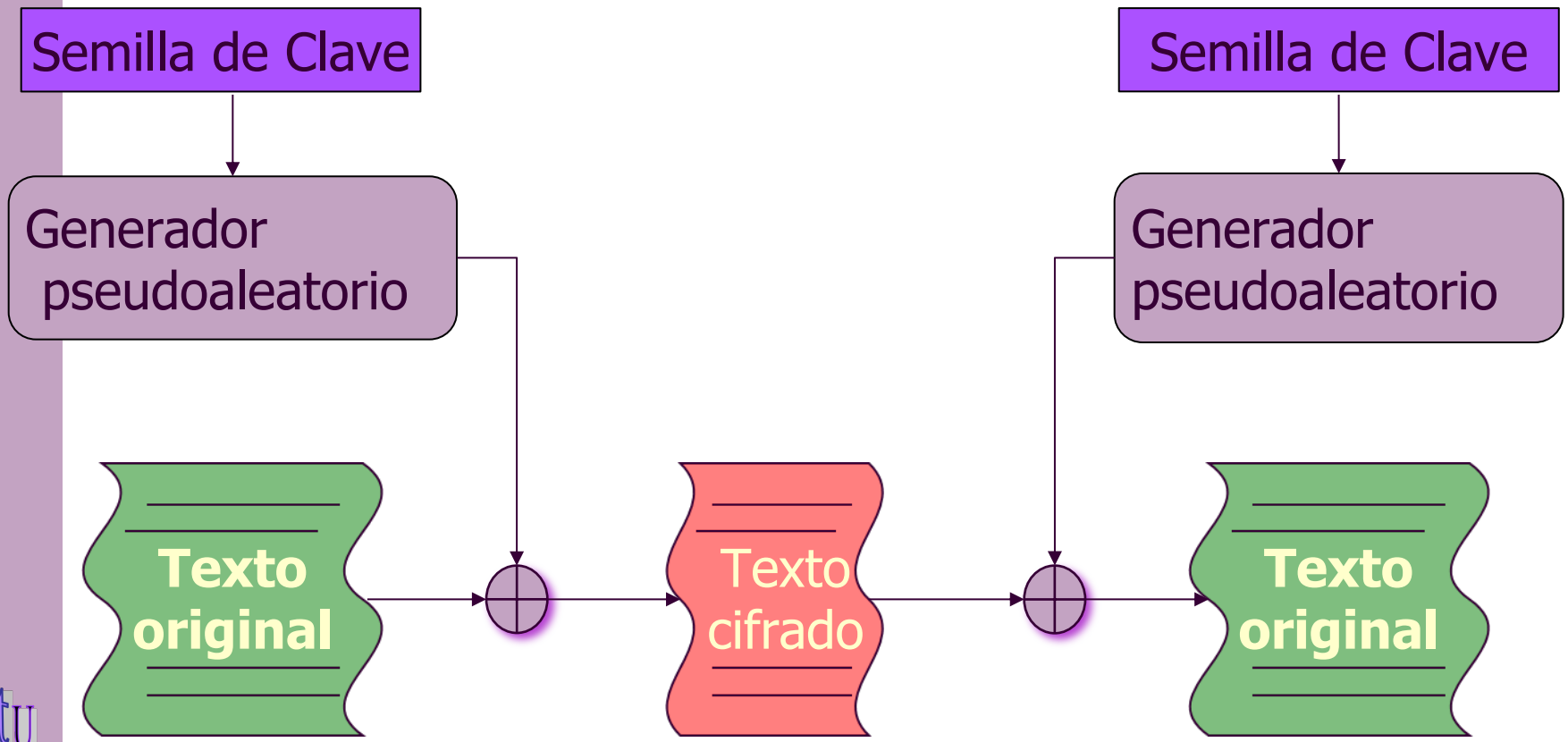
Protección de la confidencialidad



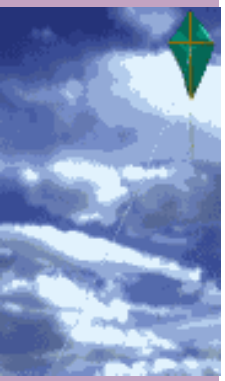
Cifrado en Flujo



Cifrado en Flujo

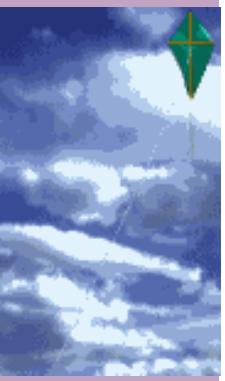


Cifrado en Flujo



- El emisor usa una semilla secreta y un algoritmo determinista (generador pseudoaleatorio) para generar una secuencia binaria.
- Con la secuencia generada y el texto en claro expresado en binario se realiza una **XOR** bit a bit.
- Realizando la misma operación con el texto cifrado y la misma secuencia pseudoaleatoria se recupera el texto en claro.

Generadores Pseudoaleatorios



1. RC4
2. ChaCha20
3. C/A
4. P(Y)
5. Snow3G
6. ZUC

P. Caballero Gil

Cifrado RC4

- Ha sido el cifrado en flujo más usado del mundo.
- Fue diseñado por Ronald Rivest en 1987.
- El algoritmo no fue publicado, pero en 1994 apareció en Internet.
- Se puede usar, pero con otro nombre: ARCFOUR, ARC4, Alleged-RC4.
- Es de implementación sencilla, y orientado a operaciones con bytes, por lo que es muy rápido en software.
- **Se ha usado en:**
  - **Wi-Fi** (WEP y WPA en estándar IEEE 802.11)
  - **TLS/SSL** (navegación segura por Internet)
  - **SSH** (intérprete para conexión a máquinas remotas)
  - **Kerberos** (servidor centralizado para autenticación en redes)
  - **Whatsapp, Skype, cifrado de pdf…**
- Algunos modos de uso lo han llevado a ser considerado **inseguro**.

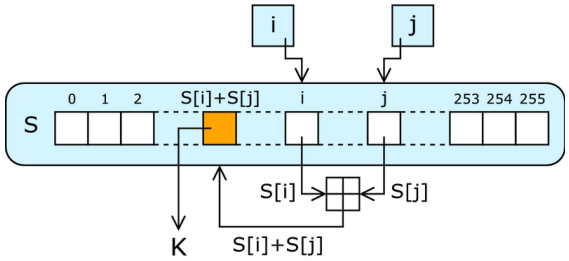
Generador RC4

- Permite claves de diferentes longitudes de 1 a 256 bytes.
- La clave secreta se usa para inicializar un vector de estado y después no se vuelve a usar.
- El vector de estado S es un array 16x16 $S= \{S_i\}_{i=0,1,\ldots,255}$ que contiene una permutación de todos los números enteros de 0 a 255 (representados cada uno con 1 byte).
- Cada vez que se cifra/descifra un byte de texto, se genera un byte de secuencia cifrante, que es un elemento S_t del vector de estado, y después se permutar dos bytes de S.
- Las permutaciones en S se hacen mediante intercambios.

Generador RC4

Inicialización (Key Scheduling Algorithm, KSA):

```
for i = 0 to 255{
   S[i] = i;
   K[i] = semilla[i mod tamañosemilla];}
j=0;
for i = 0 to 255{
   j = (j + S[i] + K[i]) mod 256;
   intercambia S[i] y S[j];}
```



Generación de secuencia cifrante (PRGA):

i = 0; j = 0;

Generar cada byte de secuencia con:

- i = (i + 1) mod 256;
- j = (j + S[i]) mod 256;
- intercambiar S[i] y S[j];
- t = (S[i] + S[j]) mod 256;
- Usar valor de S[t];

Ejemplo de RC4

Clave secreta= 1 35 69 103 137 171 205 239(Dec) =01 23 45 67 89 AB CD EF (Hex) para llenar K.

Inicializacion:

S=

00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F
…

j=(0+S[0]+K[0])=0+0+1=1 (mod 256)

Swap 01 <--> 00

01 00 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F
…

j=(1+S[1]+K[1])=1+0+35=36 (mod 256)

Swap 24 <--> 00

01 24 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F
20 21 22 23 00 25 26 27 28 29 2A 2B 2C 2D 2E 2F
…

j=(36+S[2]+K[2])=36+2+69=107 (mod 256)

Swap 6B <--> 02

01 24 6B 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F
20 21 22 23 00 25 26 27 28 29 2A 2B 2C 2D 2E 2F
30 31 32 33 34 35 36 37 38 39 3A 3B 3C 3D 3E 3F
40 41 42 43 44 45 46 47 48 49 4A 4B 4C 4D 4E 4F
50 51 52 53 54 55 56 57 58 59 5A 5B 5C 5D 5E 5F
60 61 62 63 64 65 66 67 68 69 6A 02 6C 6D 6E 6F
…

Ejemplo de RC4

S final tras inicialización:

01 38 6B 3D AC 47 E5 1B 8B 10 E8 D1 66 26 F9 E7
C9 35 7B 0E 55 4A 2A 4F E1 94 48 22 D2 21 43 60
23 15 14 EB 68 6D 33 82 F1 77 0C F8 AD 93 80 CF
C5 56 8C 92 E3 9F 40 CE 09 B8 99 C8 5A 5C 50 7D
13 **74** 4C A6 73 DD 59 0B 1C 52 41 C7 9C 19 AF ED
EF 81 27 D5 E0 54 D3 E4 07 9D FC FE 91 20 45 F7
6E 03 08 DB 61 62 FB B2 BA 9B 72 39 1E 46 70 BE
CA 25 17 CC 1F 2D 31 B6 28 B3 1D 7F 3F F5 44 8A
0D 84 BD FD B4 00 8F 58 02 5F CD BF 63 2E 6F 36
A8 34 6C 85 53 C4 71 3A 05 95 4E A5 7E 30 B1 9E
86 5E A4 AE 2C D0 CB B5 5D 1A E9 5B 8D 65 9A 11
78 C0 7A D4 06 83 B0 E2 D6 75 37 4B 49 DC DF 8E
16 69 29 18 76 AA 67 EA A9 24 A3 D9 98 2F 3B D8
79 12 EE C2 B9 F4 E6 F6 3E BC BB 3C A0 88 DE 57
96 87 C1 04 DA C6 97 0A 2B 51 4D D7 B7 7C F3 FF
F0 F2 AB FA 64 90 EC C3 89 A2 32 A1 42 A7 0F 6A

Generación de 1º byte de secuencia cifrante:

$i=j=0,\ i = (i + 1) \equiv 1,\ j = (j + S[01]) = 38_{(hx}$

swap S[01] <--> S[38]

09 <--> 38

$t = (S[01] + S[38]) \equiv 09_{(hx} + 38_{(hx} = 41_{(hx}$

1º byte de secuencia cifrante=$S[41]=74_{(hx}$ = **0111 0100**

S tras 1º byte se secuencia cifrante:

01 09 6B 3D AC 47 E5 1B 8B 10 E8 D1 66 26 F9 E7
C9 35 7B 0E 55 4A 2A 4F E1 **94** 48 22 D2 21 43 60
23 15 14 EB 68 6D 33 82 F1 77 0C F8 AD 93 80 CF
C5 56 8C 92 E3 9F 40 CE 38 B8 99 C8 5A 5C 50 7D
13 74 4C A6 73 DD 59 0B 1C 52 41 C7 9C 19 AF ED
EF 81 27 D5 E0 54 D3 E4 07 9D FC FE 91 20 45 F7
6E 03 08 DB 61 62 FB B2 BA 9B 72 39 1E 46 70 BE
CA 25 17 CC 1F 2D 31 B6 28 B3 1D 7F 3F F5 44 8A
0D 84 BD FD B4 00 8F 58 02 5F CD BF 63 2E 6F 36
A8 34 6C 85 53 C4 71 3A 05 95 4E A5 7E 30 B1 9E
86 5E A4 AE 2C D0 CB B5 5D 1A E9 5B 8D 65 9A 11
78 C0 7A D4 06 83 B0 E2 D6 75 37 4B 49 DC DF 8E
16 69 29 18 76 AA 67 EA A9 24 A3 D9 98 2F 3B D8
79 12 EE C2 B9 F4 E6 F6 3E BC BB 3C A0 88 DE 57
96 87 C1 04 DA C6 97 0A 2B 51 4D D7 B7 7C F3 FF
F0 F2 AB FA 64 90 EC C3 89 A2 32 A1 42 A7 0F 6A

Generación de 2º byte de secuencia cifrante:

$i = (i + 1) \equiv 2,\ j = (j + S[02]) = 38_{(hx} + 6B_{(hx} = A3_{(hx}$

swap S[02] <--> S[A3]

6B <--> AE

$t = (S[02]+S[A3]) \equiv 6B_{(hx} + AE_{(hx} \equiv 19_{(hx} \pmod{256}$

2º byte de secuencia cifrante=$S[19]=94_{(hx}$ =**1001 0100**

https://youtu.be/kfdvlaOD1ig

Ataque NoMore a RC4

En 2015 se publicó un ataque práctico contra el algoritmo RC4 llamado **Ataque NoMore**, que:

- contra WPA-TKIP permite descifrar e inyectar paquetes arbitrarios en una hora, y
- contra TLS permite descifrar una cookie HTTPs en 75 horas.

https://www.elladodelmal.com/2015/07/rc4-no-more-robo-de-cookies-en-httpstls.html

Cifrados en Wi-Fi

- **WEP** (RC4) desde 1997
- **WPA** (RC4) desde 2003
- **WPA2** (AES y RC4) desde 2006
- **WPA3** (AES) desde 2018 **en WiFi 6**

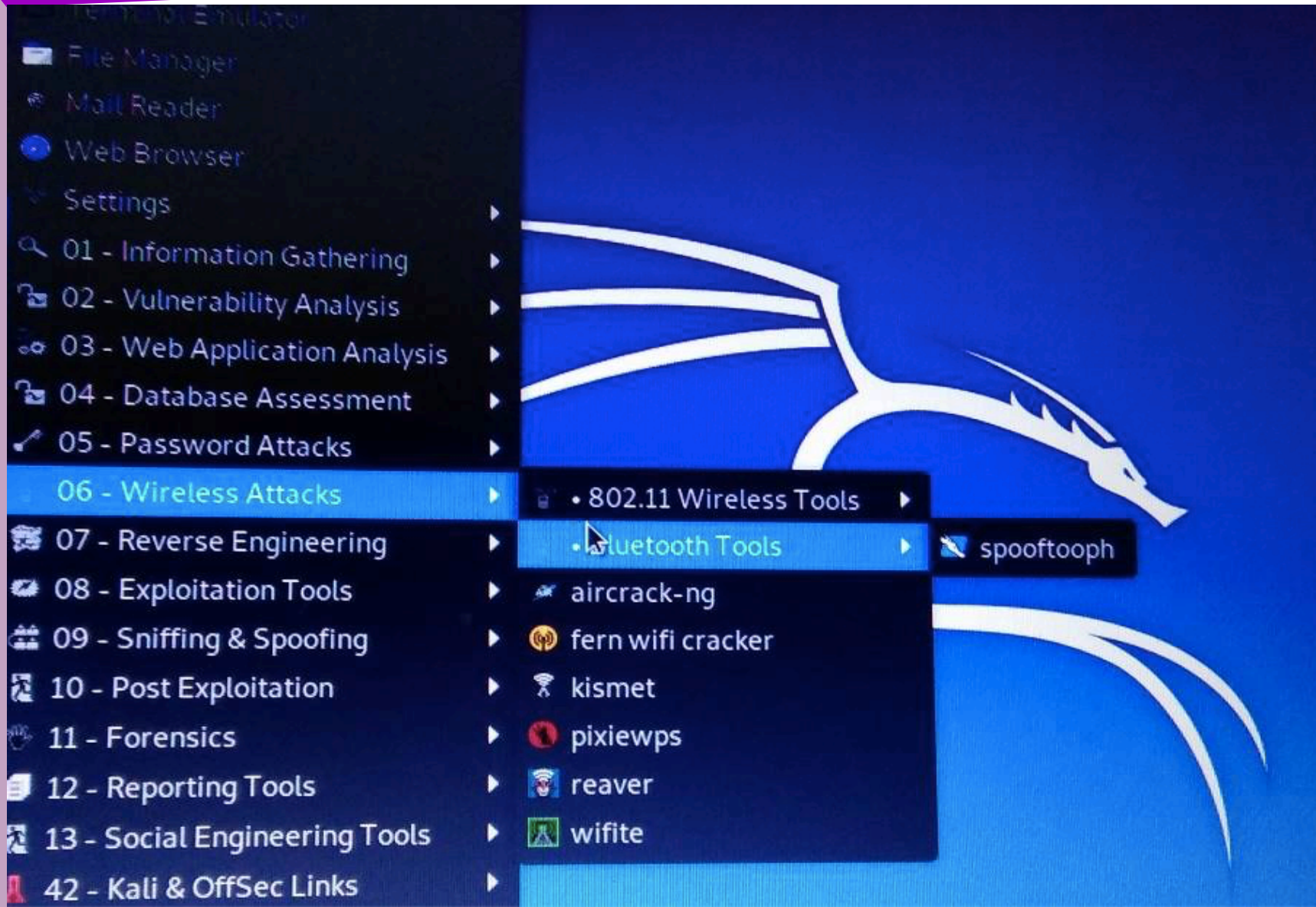
Inseguridad de WPA2

- WPA2 (según standard IEEE802.11i): Obligatorio desde 2006.
- Solo cambió el cifrado. La autenticación es igual que en WPA.
- **WPA2-PSK (AES)** **es la opción más segura** pues garantiza el uso del cifrado en bloque Rijndael (AES).
- WPA2-PSK (TKIP) es un opción insegura, que usa RC4.
- Durante el intercambio de información en el proceso de conexión, si el cliente no soporta las autenticaciones que especifica el AP, es desconectado, pudiendo sufrir un ataque DoS.
- También se puede romper el handshake en la autenticación.
- Es conveniente desactivar WPS (Wi-Fi Protected Setup) para conexión con PIN sin contraseña.

Ataque KRACK a WPA2

- **En 2017 se publicó el ataque KRACK** (Key Reinstallation AttaCK) para romper WPA2 basándose en una debilidad del estándar.
- Es un ataque de **reproducción** o reinyección en el que una transmisión interceptada es maliciosamente repetida para forzar la **reinstalación de la clave** compartida, con objeto de descifrar luego los datos que transmite la víctima.
- Ataca el **handshake** que se realiza cuando un usuario se conecta a la red, cuando se verifican las credenciales de cliente y punto de acceso. Una vez acabado el handshake, se genera una clave de cifrado compartida que se usa para cifrar todo el tráfico.
- Se trata de engañar a la víctima para que reinstale una clave usada.
- El ataque es peor si se usa **WPA2-PSK (TKIP) o AES-GCMP**, ya que permite no sólo descifrar tráfico sino inyectar paquetes.
- El handshake de **WPA3** es resistente al ataque KRACK.

Kali Linux y Aircrack



P. Caballero Gil

Cifrado en WPA3 de WiFi 6

- **WPA3 llegó en 2018 para reemplazar a WPA2**.
- Las redes WPA3 pueden impedir que los dispositivos que solo soportan WPA2 se conecten.
- Usa el cifrado AES **con claves de longitud 192 bits en vez de 128 bits.**
- WPA3 es más seguro aunque la clave de la conexión Wi-Fi que escoja el usuario no sea segura.
- Incluye Wi-Fi Easy Connect para configurar y conectar dispositivos IoT sin pantalla ni botones físicos.

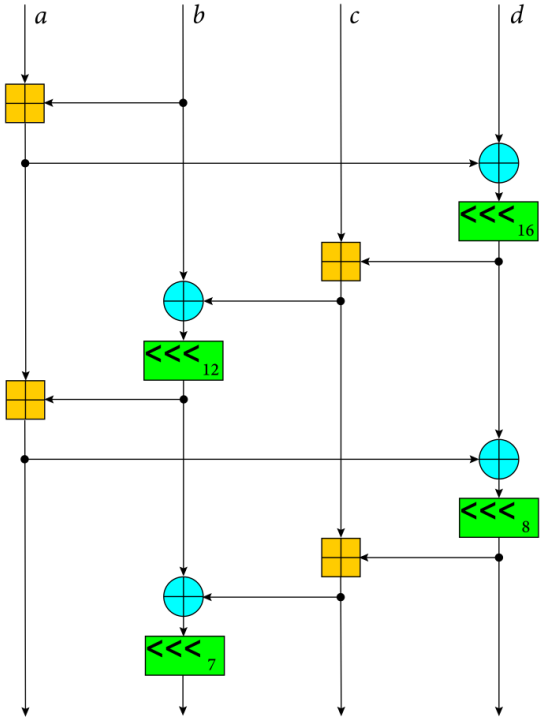
P. Caballero Gil

Cifrado en SSL/TLS

Cifrado			Versión del Protocolo					
Tipo	Algoritmo	Fortaleza nominal (bits)	SSL 2.0	SSL 3.0 note 1 note 2 note 3	TLS 1.0 note 1 note 3	TLS 1.1 note 1	TLS 1.2 note 1	TLS 1.3[14]
Cifrado por bloques	AES GCM[15] note 4	256, 128	N/D	N/D	N/D	N/D	Seguro	Seguro
	AES CCM[16] note 4		N/D	N/D	N/D	N/D	Seguro	Seguro
	AES CBC note 5		N/D	N/D	depende	depende	depende	N/D
	Camellia GCM[17] note 4	256, 128	N/D	N/D	N/D	N/D	Seguro	N/D
	Camellia CBC[18] note 5		N/D	N/D	depende	depende	depende	N/D
	ARIA GCM	256, 128	N/D	N/D	N/D	N/D	Seguro	N/D
	ARIA CBC		N/D	N/D	depende	depende	depende	N/D
	SEED CBC[19] note 5	128	N/D	N/D	depende de las mitigaciones	depende de las mitigaciones	depende de las mitigaciones	N/D
	3DES EDE CBC note 5 note 6	112 note 7	Inseguro	Inseguro	Inseguro	Inseguro	Inseguro	N/D
	GOST 28147-89 CNT[13] note 6	256	N/D	N/D	Inseguro	Inseguro	Inseguro	N/D
	IDEA CBC note 5 note 6 note 8	128	Inseguro	Inseguro	Inseguro	Inseguro	N/D	N/D
	DES CBC note 5 note 6 note 8	56	Inseguro	Inseguro	Inseguro	Inseguro	N/D	N/D
		40 note 9	Inseguro	Inseguro	Inseguro	N/D	N/D	N/D
	RC2 CBC note 5 note 6	40 note 9	Inseguro	Inseguro	Inseguro	N/D	N/D	N/D
Cifrador de flujo	ChaCha20+Poly1305[24] note 4	256	N/D	N/D	N/D	N/D	Seguro	Seguro
	RC4 note 10	128	Inseguro	Inseguro	Inseguro	Inseguro	Inseguro	N/D
		40	Inseguro	Inseguro	Inseguro	N/D	N/D	N/D

Cifrado ChaCha20 de SSL/TLS

- Cifrado en flujo desarrollado en 2008 a partir del cifrado Salsa20 enviado a eSTREAM por Daniel **Bernstein**.
- Se basa en un generador pseudoaleatorio definido sobre operaciones **ARX: Agregar-Rotar-XOR**, consistentes en sumas de 32 bits (mod 2^{32}), sumas de bits (XOR) y rotaciones.
- El generador original se alimenta de una clave de **256 bits**, un nonce pseudoaleatorio de 64 bits y un contador de 64 bits, y produce una secuencia cifrante de 512 bits.
- Se suele combinar con **Poly1305**, que es un código de autenticación de mensaje.
- Hay implementaciones de dominio público.



Cifrado ChaCha20 de SSL/TLS

- Las operaciones se realizan sobre un **estado interno de 512 bits, e.d. 16 palabras de 32 bits dispuestas como matrices 4×4**:

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

- En el cifrado original el **estado inicial** se compone de: una clave de 256 bits, un contador de 64 bits, un nonce de 64 bits, y una constante de 128 bits, organizadas como: 8 palabras de clave, 2 palabras contador de posición, 2 palabras de nonce pseudoaleatorio y 4 palabras fijas:

"expa"	"nd 3"	"2-by"	"te k"
Key	Key	Key	Key
Key	Key	Key	Key
Pos.	Pos.	Nonce	Nonce

- La operación principal es **QR(a,b,c,d)**, que toma una entrada de 4 palabras, y la actualiza como salida de 4 palabras de forma que:

```
a += b; d ^= a; d <<<= 16;
c += d; b ^= c; b <<<= 12;
a += b; d ^= a; d <<<= 8;
c += d; b ^= c; b <<<= 7;
```

- Hay 20 iteraciones de QR(a,b,c,d), t.q. en las
  - **impares** se aplica sobre las 4 **columnas**
  - **pares** se aplica sobre las 4 **diagonales** (filas en Salsa20)

Implementaciones del Cifrado ChaCha20

Algorithm 1 ChaCha20 Stream Cipher:
$C = \text{ChaCha20-SC}(K, N, P)$.

Input: $K \in \{0,1\}^{256}$, $N \in \{0,1\}^{96}$, $P \in \{0,1\}^*$
Output: $C \in \{0,1\}^{|P|}$

for $i \leftarrow 0$ **to** $\lceil |P|/512 \rceil - 1$ **do**
  /* Init State */
  $S[0] \leftarrow$ 0x61707865, $S[1] \leftarrow$ 0x3320646e
  $S[2] \leftarrow$ 0x79622d32, $S[3] \leftarrow$ 0x6b206574
  $S[4..11] \leftarrow K$ {Set Key}
  $S[12] \leftarrow i$ {Set Counter}
  $S[13..15] \leftarrow N$ {Set Nonce}
  $S' \leftarrow S$ {Save Initial State}
  for $n \leftarrow 0$ **to** 9 **do** {10 Double Rounds}
    /* Column Round */
    $S[0,4,8,12] \leftarrow \text{QR}(S[0], S[4], S[8], S[12])$
    $S[1,5,9,13] \leftarrow \text{QR}(S[1], S[5], S[9], S[13])$
    $S[2,6,10,14] \leftarrow \text{QR}(S[2], S[6], S[10], S[14])$
    $S[3,7,11,15] \leftarrow \text{QR}(S[3], S[7], S[11], S[15])$
    /* Diagonal Round */
    $S[0,5,10,15] \leftarrow \text{QR}(S[0], S[5], S[10], S[15])$
    $S[1,6,11,12] \leftarrow \text{QR}(S[1], S[6], S[11], S[12])$
    $S[2,7,8,13] \leftarrow \text{QR}(S[2], S[7], S[8], S[13])$
    $S[3,4,9,14] \leftarrow \text{QR}(S[3], S[4], S[9], S[14])$
  end for
  $k_i^N \leftarrow S \boxplus S'$ $\{\forall 0 \le x \le 15 : S[x] \boxplus S'[x]\}$
  $c_i \leftarrow p_i \oplus k_i^N$ {Encrypt}
end for
return C

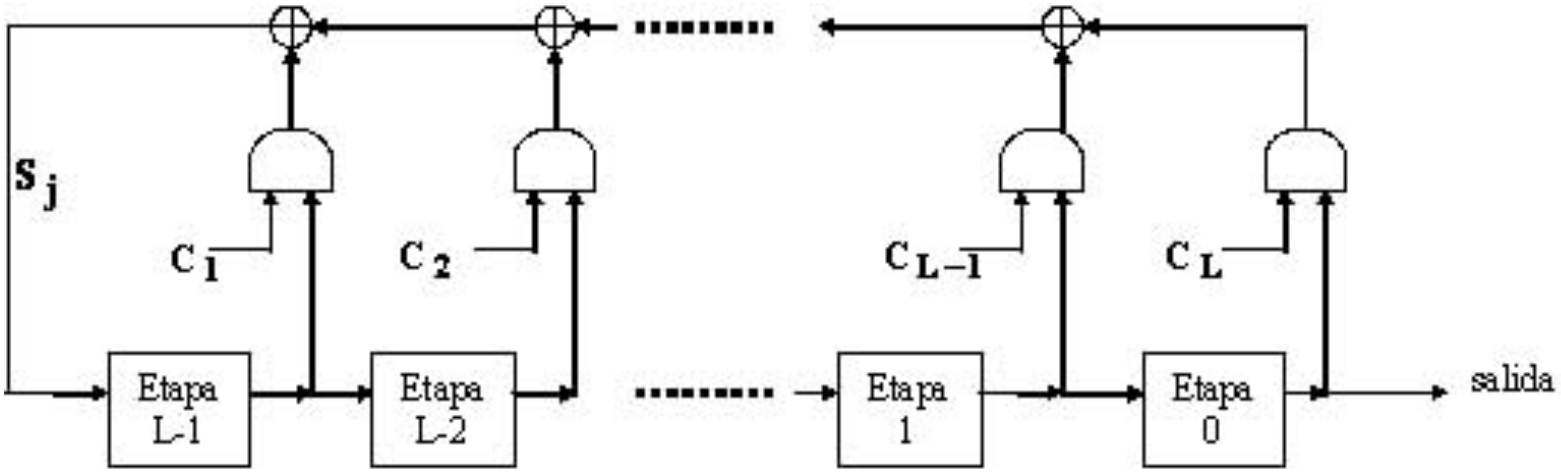
```c
#define ROTL(a,b) (((a) << (b)) | ((a) >> (32 - (b))))
#define QR(a, b, c, d) (            \
    a += b,  d ^= a,  d = ROTL(d,16),   \
    c += d,  b ^= c,  b = ROTL(b,12),   \
    a += b,  d ^= a,  d = ROTL(d, 8),   \
    c += d,  b ^= c,  b = ROTL(b, 7))
#define ROUNDS 20

void chacha_block(uint32_t out[16], uint32_t const in[16])
{
    int i;
    uint32_t x[16];

    for (i = 0; i < 16; ++i)
        x[i] = in[i];
    // 10 loops × 2 rounds/loop = 20 rounds
    for (i = 0; i < ROUNDS; i += 2) {
        // Odd round
        QR(x[0], x[4], x[ 8], x[12]); // column 0
        QR(x[1], x[5], x[ 9], x[13]); // column 1
        QR(x[2], x[6], x[10], x[14]); // column 2
        QR(x[3], x[7], x[11], x[15]); // column 3
        // Even round
        QR(x[0], x[5], x[10], x[15]); // diagonal 1 (main diagonal)
        QR(x[1], x[6], x[11], x[12]); // diagonal 2
        QR(x[2], x[7], x[ 8], x[13]); // diagonal 3
        QR(x[3], x[4], x[ 9], x[14]); // diagonal 4
    }
    for (i = 0; i < 16; ++i)
        out[i] = x[i] + in[i];
}
```

https://tools.ietf.org/html/rfc7539 (donde se usa un contador de 32 bits y un nonce de 96 bits)

Registro de Desplazamiento con Realimentación Lineal

•En GPS, Bluetooth, telefonía móvil y cifrados militares se usan cifrados en flujo basados en RDRLs o Linear Feedback Shift Registers (LFSRs)

•Generador de secuencia binaria pseudoaleatoria s_j.



•Cada etapa de la estructura se inicializa con un bit.

•A cada golpe de reloj, se realimenta con un bit producido mediante XORs definidos por la estructura.

RDRL

Polinomio de realimentación:

$C(x)=1+c_1x+c_2x^2+\ldots+c_Lx^L$ con c_i binarios

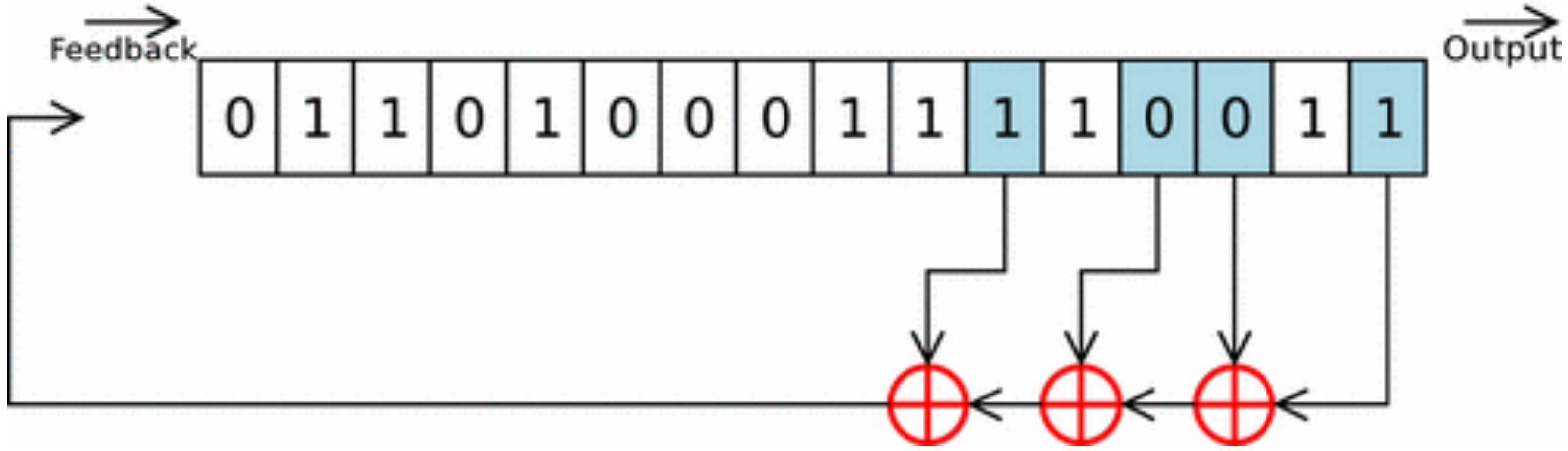
•El polinomio de realimentación

determina el periodo de la secuencia.

•El periodo máximo es 2^L-1

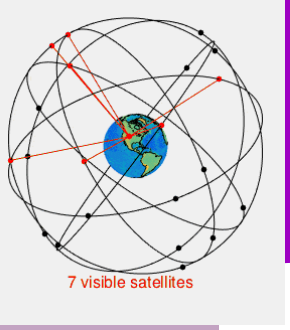
m	Polinomios primitivos	m	Polinomios primitivos
1	$x+1$	13	$x^{13}+x^4+x^3+x+1$
2	x^2+x+1	14	$x^{14}+x^{10}+x^6+x+1$
3	x^3+x+1	15	$x^{15}+x+1$
4	x^4+x+1	16	$x^{16}+x^{12}+x^3+x+1$
5	x^5+x^2+1	17	$x^{17}+x^3+1$
6	x^6+x+1	18	$x^{18}+x^7+1$
7	x^7+x^3+1	19	$x^{19}+x^5+x^2+x+1$
8	$x^8+x^4+x^3+x^2+1$	20	$x^{20}+x^3+1$
9	x^9+x^4+1	21	$x^{21}+x^2+1$
10	$x^{10}+x^3+1$	22	$x^{22}+x+1$
11	$x^{11}+x^2+1$	23	$x^{23}+x^5+1$
12	$x^{12}+x^6+x^4+x+1$	24	$x^{24}+x^7+x^2+x+1$

(Polinomio primitivo: irreducible cuyas raíces son elementos primitivos, e.d. con potencias que generan todo el cuerpo de Galois)



Ej: polinomio de realimentación: $1 + x^{11} + x^{13} + x^{14} + x^{16}$

P. Caballero Gil



Comunicaciones en GPS

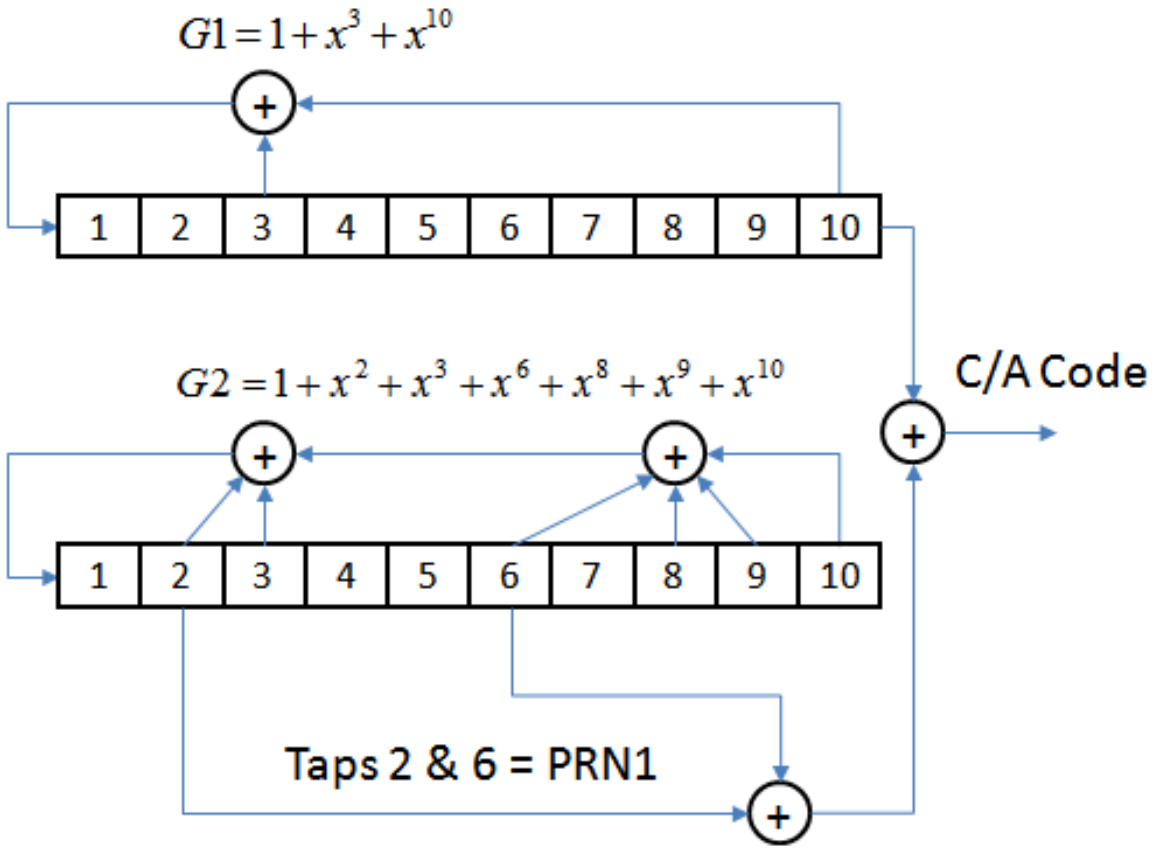
1. **Código de Adquisición Aproximativa C/A**: Generador que se usa en el servicio estándar de posicionamiento para uso civil. Genera una secuencia de periodo 2^{10}-1=1023 que se repite cada milisegundo, y que es diferente para cada satélite.
2. **Cifrado de Precisión P(Y)**: Generador que se usa para posicionamiento de seguridad con fines militares. Es un cifrado en flujo con secuencia cifrante de periodo $2{,}35*10^{14}$ que se repite cada 266 días (con reset cada domingo), y en el que cada satélite usa un fragmento diferente de la secuencia.

- C/A utiliza 2 LFSRs de longitud 10 y estados iniciales 111…..1.
- P(Y) utiliza 4 LFSRs de longitud 12.
- Las secuencias generadas en C/A y P(Y) son secuencias de Gold pues salen de la suma de las salidas de LFSRs paralelos.

Generador C/A de GPS

Las secuencias de Gold tienen baja correlación cruzada con otras generadas con el mismo generador, lo que es útil cuando varios dispositivos transmiten en el mismo rango de frecuencia.

PRN ID	G2 Taps	PRN ID	G2 Taps
1	2 & 6	17	1 & 4
2	3 & 7	18	2 & 5
3	4 & 8	19	3 & 6
4	5 & 9	20	4 & 7
5	1 & 9	21	5 & 8
6	2 & 10	22	6 & 9
7	1 & 8	23	1 & 3
8	2 & 9	24	4 & 6
9	3 & 10	25	5 & 7
10	2 & 3	26	6 & 8
11	3 & 4	27	7 & 9
12	5 & 6	28	8 & 10
13	6 & 7	29	1 & 6
14	7 & 8	30	2 & 7
15	8 & 9	31	3 & 8
16	9 & 10	32	4 & 9



(El par de celdas del 2º RDRL depende del satélite)

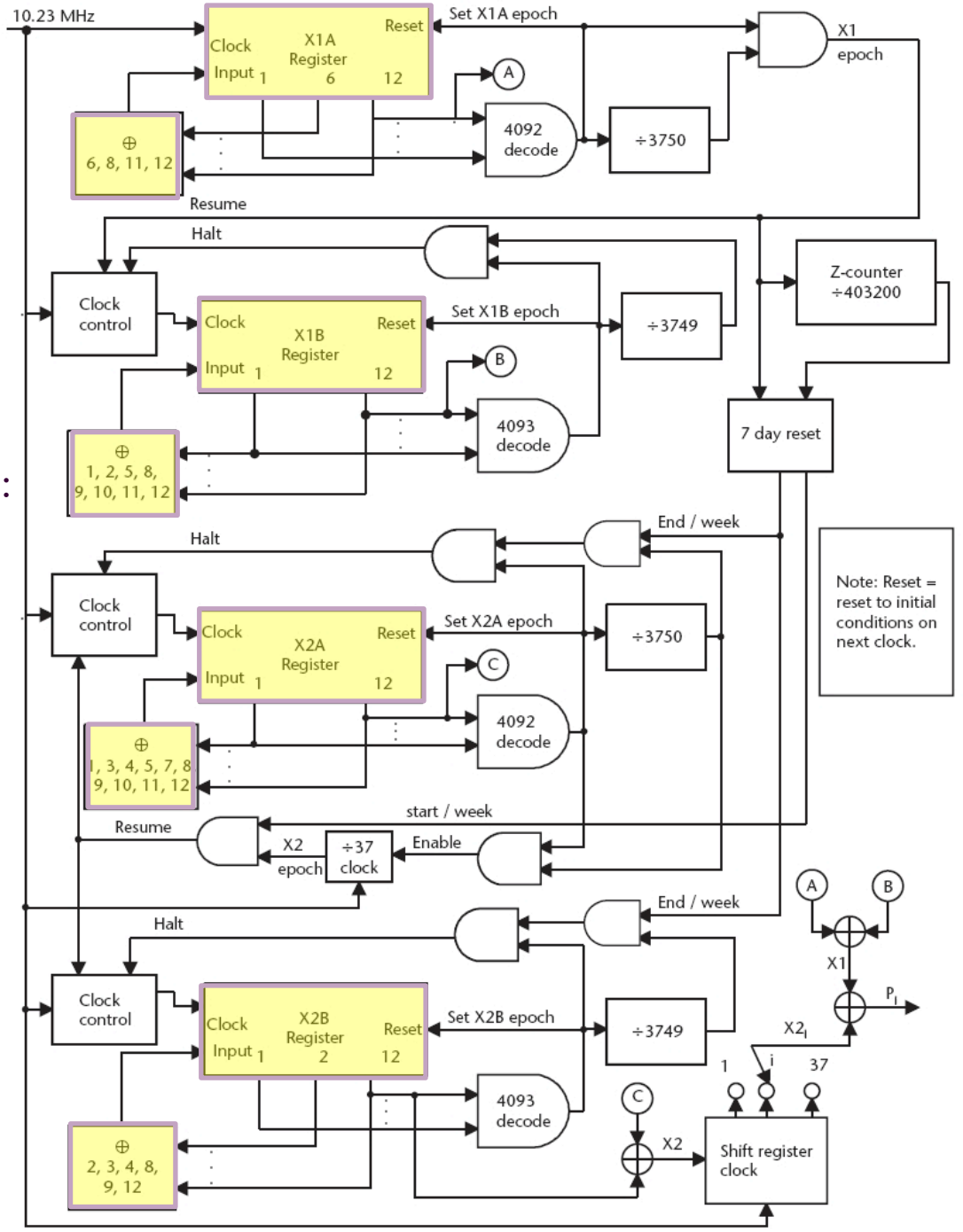
Cifrado P(Y) de GPS

GPS

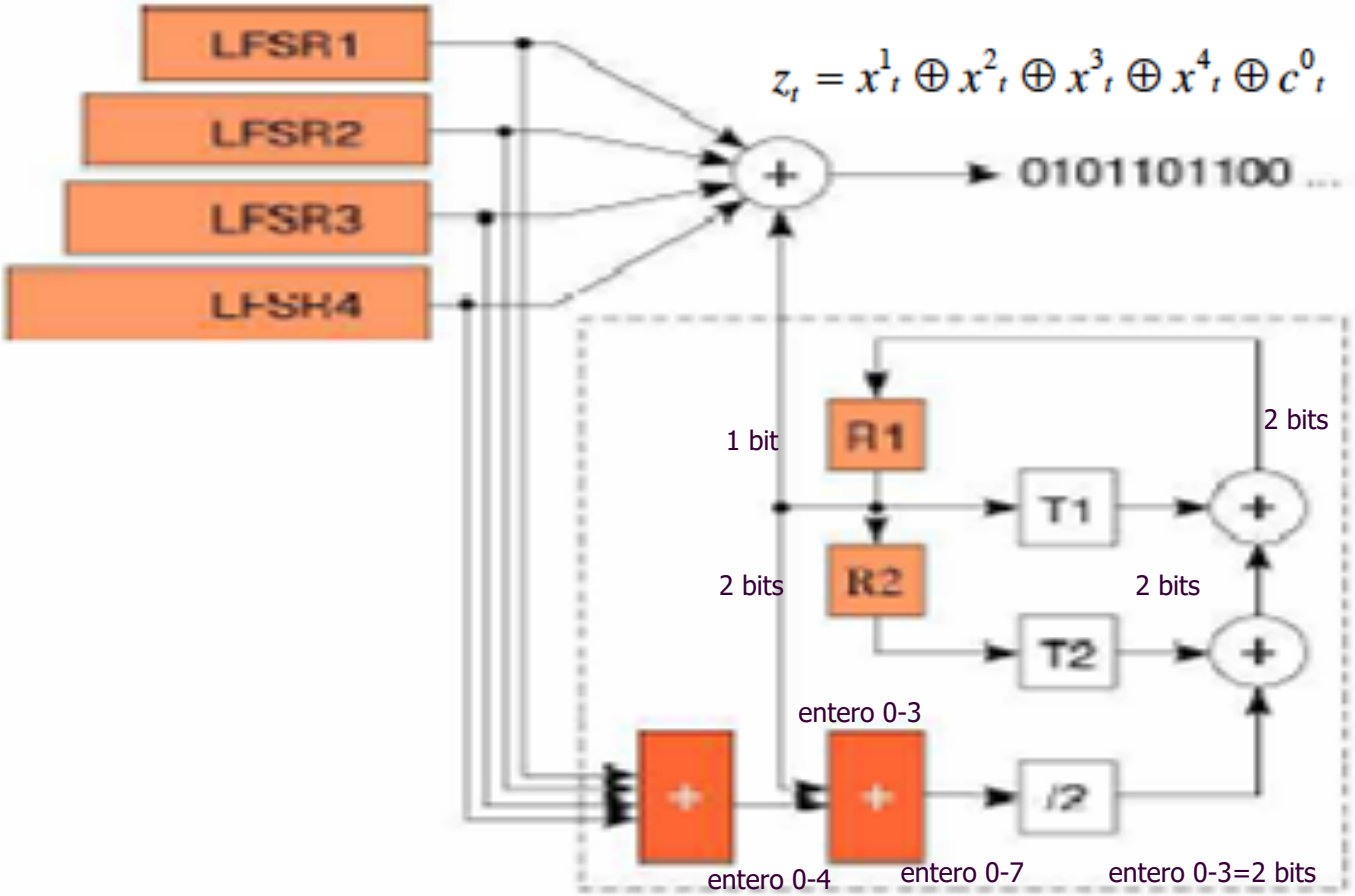
Estados iniciales de los RDRLs:

- De X1A: 001001001000
- De X1B: 010101010100
- De X2A: 100100100101
- De X2B: 010101010100

https://www.gps.gov/technical/icwg/IS-GPS-200K.pdf



Generador E0 de Bluetooth



$$T_1(C_t^1, C_t^0) = (C_t^1, C_t^0)$$

$$T_2(C_{t+1}^1, C_{t+1}^0) = (C_{t+1}^0, C_{t+1}^1 \oplus C_{t+1}^0)$$

Cifrado en BLE y NFC

Bluetooth Low Energy (BLE)

- La versión anterior de Bluetooth usaba un cifrado en flujo llamado E0 basado en 4 RDRLs.
- La seguridad de **BLE se basa en el cifrado AES.**
- BLE fue diseñado para que funcione con poca energía, bajo coste, cobertura de hasta 50m y pequeño tamaño.

Near Field Communication (NFC)

- Tecnología de comunicación de muy corto alcance, hasta 0.1m, de forma que gran parte de su seguridad se basa en la corta distancia que requiere su uso.
- **NFC no cifra las comunicaciones por defecto,** aunque la mayoría de aplicaciones que lo usan sí incorporan cifrado.

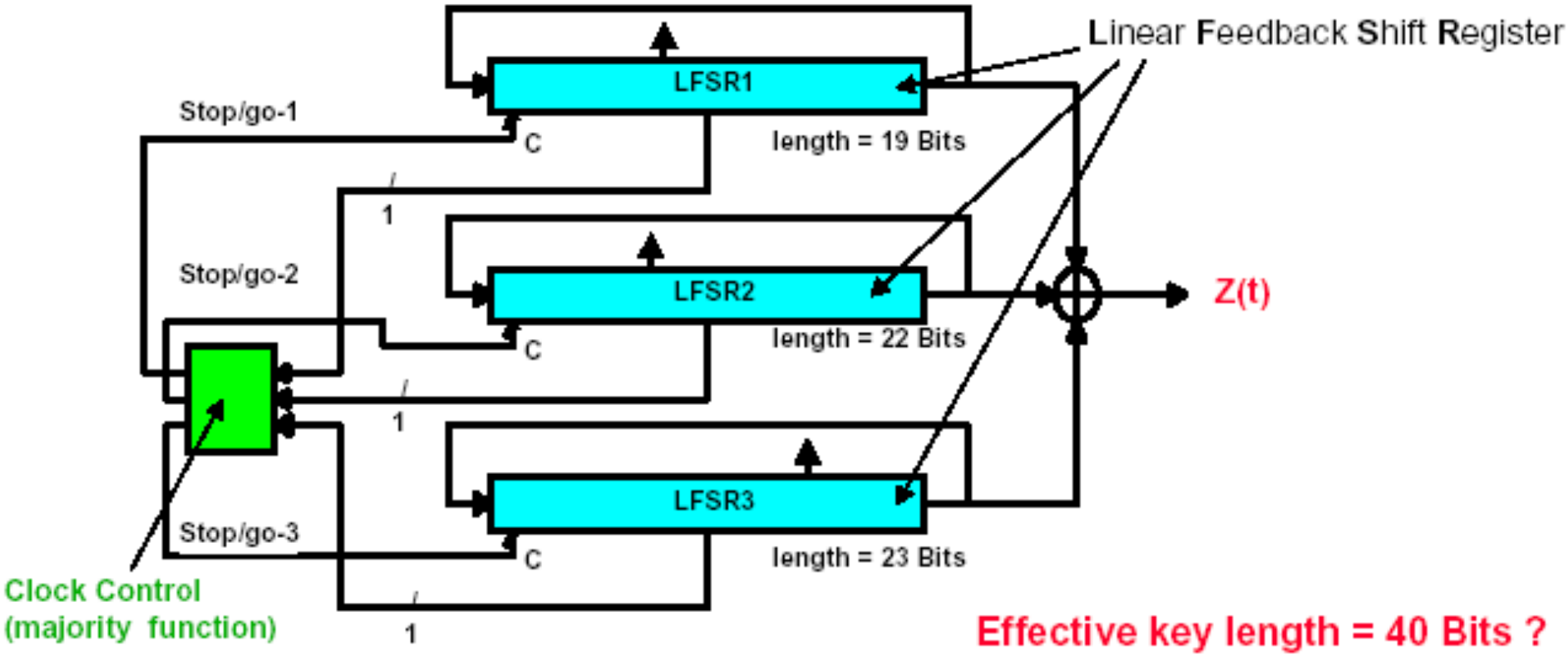
Cifrado en Telefonía Móvil

- 2G o GSM (**A5**) desde 1999
- 3G o UMTS (**Kasumi y SNOW3G**) desde 2002
- 4G o LTE (**SNOW3G y AES**) desde 2010
- 5G (**SNOW3G, ZUC y AES**) desde 2020

movistar vodafone orange™ yoigo

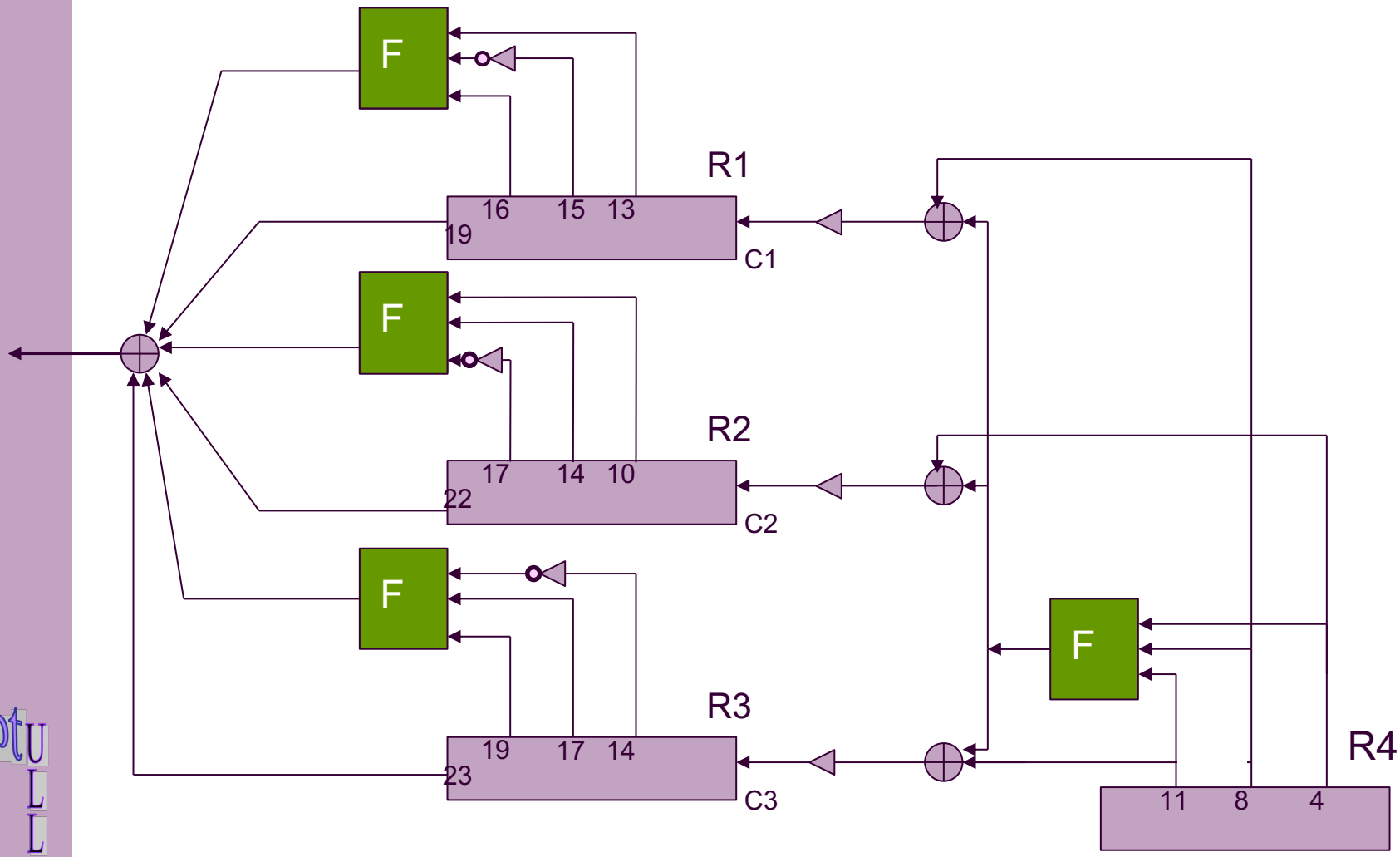
http://www.movistar.es/particulares/coberturas/movil/5G/

Generador A5/1 de GSM



P. Caballero Gil

Generador A5/2 de GSM



P. Caballero Gil

Cifrado en 3G (UMTS), 4G (LTE) y 5G

Para confidencialidad en UMTS:

- UEA1 = KASUMI
- UEA2 = **SNOW 3G**

Para confidencialidad en LTE:

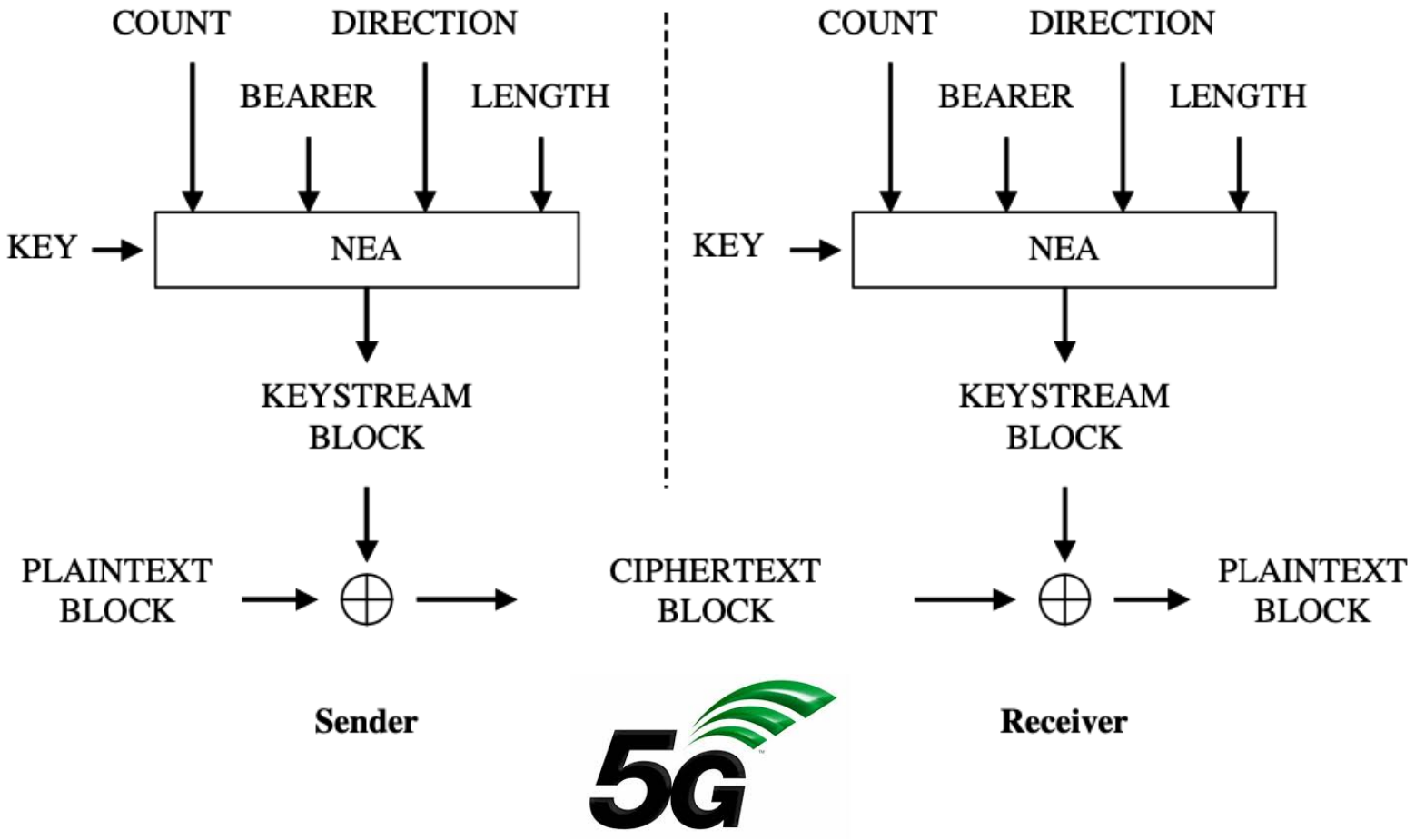
- 128-EEA1 = **SNOW 3G**
- 128-EEA2 = AES
- 128-EEA3 = **ZUC**

Para confidencialidad e integridad:

- NEA1 = NIA1 = **SNOW 3G**
- NEA2 = NIA2 = AES
- NEA3 = NIA3 = **ZUC**

Cifrado en 5G

Entrada: Clave de 128 bits, Contador de 32 bits, Identidad del portador de 5 bits, Dirección de la transmisión de 1 bit, y Longitud de la secuencia cifrante necesaria.



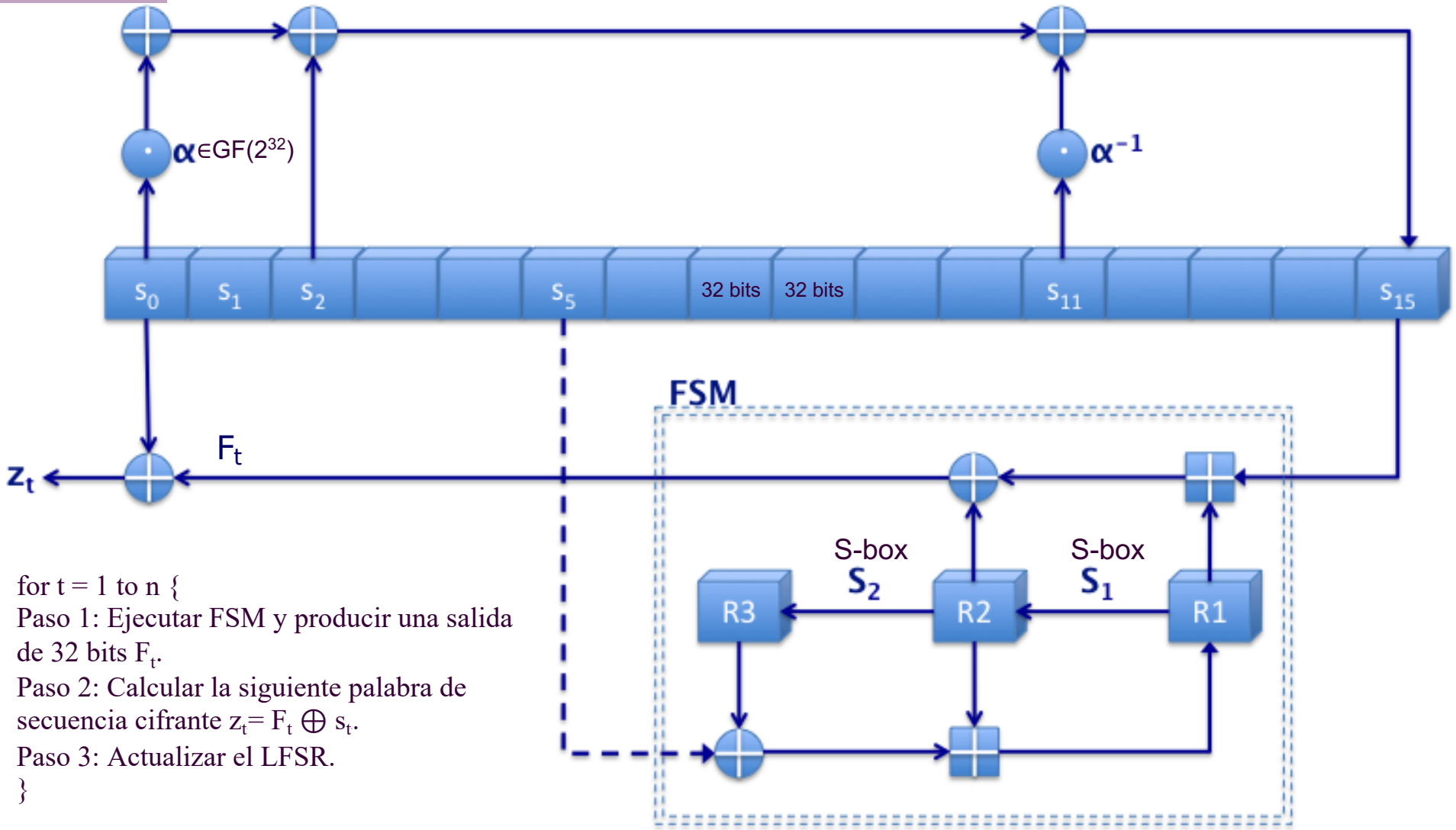
https://www.etsi.org/deliver/etsi_ts/133500_133599/133501/15.02.00_60/ts_133501v150200p.pdf

Generador SNOW 3G de 3G, 4G y 5G

- SNOW 3G consta de dos partes:
  - un LFSR y
  - una máquina de estados finitos o FSM (Finite State Machine).
- El LFSR tiene longitud 16 y **cada elemento s_0, s_1, s_2,..., s_{15}, contiene 4 bytes, e.d. 32 bits**.
- La realimentación se define con un polinomio primitivo sobre un cuerpo extendido GF(2^{32}), e implica dos multiplicaciones, una por una constante α de GF(2^{32}) y otra por la inversa de α.

$$s_{t+16} = \alpha s_t \oplus s_{t+2} \oplus \alpha^{-1} s_{t+11}, \forall t \geq 0$$

Generador SNOW 3G de 3G, 4G y 5G



for t = 1 to n {
Paso 1: Ejecutar FSM y producir una salida de 32 bits F_t.
Paso 2: Calcular la siguiente palabra de secuencia cifrante $z_t = F_t \oplus s_t$.
Paso 3: Actualizar el LFSR.
}

SNOW 3G

- La **FSM** es la parte no lineal del generador y se alimenta de dos valores de entrada procedentes del LFSR correspondientes a los elementos en las posiciones 5 y 15.
- La FSM está formada por tres registros de 32 bits, R1, R2 y R3, y dos cajas de sustitución o S-boxes, S1 y S2, que se usan para actualizar los registros R2 y R3.
- En la FSM, las **S-Boxes** S1 y S2 toman 32 bits y producen 32 bits tras varias combinaciones de una S-box básica sobre cada uno de los 4 bytes de entrada. La caja S1 se basa en la S-box usada en el cifrado AES, mientras que la S-box S2 fue diseñada para SNOW 3G.
- En la FSM, las operaciones de mezcla de la FSM consisten en una **XOR bit a bit, y una suma de enteros módulo 2^{32}**.

SNOW 3G

- El LFSR puede usarse en dos modos de operación:
  - de inicialización (desechando la salida de 32 iteraciones)
  - de generación de secuencia cifrante.
- SNOW 3G se inicializa usando:
  - una clave de 128 bits como 4 palabras de 32 bits: k_0, k_1, k_2, k_3
  - un vector de inicialización IV de 128 bits como 4 palabras de 32 bits: IV_0, IV_1, IV_2, IV_3
  - Una palabra 1 de todo unos: 0xffffffff
  - R1 = R2 = R3 = 0

$$s_{15} = \mathbf{k_3} \oplus \mathbf{IV_0} \qquad s_{14} = \mathbf{k_2} \qquad s_{13} = \mathbf{k_1} \qquad s_{12} = \mathbf{k_0} \oplus \mathbf{IV_1}$$

$$s_{11} = \mathbf{k_3} \oplus \mathbf{1} \qquad s_{10} = \mathbf{k_2} \oplus \mathbf{1} \oplus \mathbf{IV_2} \qquad s_{9} = \mathbf{k_1} \oplus \mathbf{1} \oplus \mathbf{IV_3} \qquad s_{8} = \mathbf{k_0} \oplus \mathbf{1}$$

$$s_{7} = \mathbf{k_3} \qquad s_{6} = \mathbf{k_2} \qquad s_{5} = \mathbf{k_1} \qquad s_{4} = \mathbf{k_0}$$

$$s_{3} = \mathbf{k_3} \oplus \mathbf{1} \qquad s_{2} = \mathbf{k_2} \oplus \mathbf{1} \qquad s_{1} = \mathbf{k_1} \oplus \mathbf{1} \qquad s_{0} = \mathbf{k_0} \oplus \mathbf{1}$$

https://www.gsma.com/security/wp-content/uploads/2019/05/snow3gspec.pdf

Implementación de SNOW 3G

- Las dos multiplicaciones implicadas en el LFSR se corresponden con productos de polinomios en módulo $x^8+x^7+x^5+x^3+1$, pero pueden implementarse como desplazamientos de bytes, y XORs con el byte $A9_{16}=10101001_2$.
- En particular, implican aplicar operación distributiva sobre los dos bytes multiplicandos, usando para ello el byte de menor peso, y luego para cada bit 1 de ese byte, desplazar a izquierda el otro byte, de forma que cada vez que su bit más significativo antes del desplazamiento sea 1, hay que hacer, tras el desplazamiento, una XOR bit a bit con el byte $A9_{16}=10101001_2$.
- El nº de desplazamientos a realizar viene dado por la posición de los bits iguales a 1 en el 1er byte multiplicando.

Multiplicación en SNOW 3G

Los bytes se representan en forma de polinomios.

Ej: $57_{(16} = 0101\ 0111_{(2} = x^6+x^4+x^2+x+1$ $\quad$ $83_{(16} = 1000\ 0011_{(2} = x^7+x+1$

Multiplicación en GF(2^8)

Consiste en multiplicar los polinomios (en módulo 2), y el resultado reducirlo módulo el polinomio **$x^8+x^7+x^5+x^3+1$** (que se corresponde con el byte **10101001=A9**).

Ej: {57} . {83}

$$(5\ 7)_{16} \otimes (8\ 3)_{16} = (C\ 1)_{16} \rightarrow$$
$$(x^6 + x^4 + x^2 + x\ \ +1) \otimes (x^7 + x\ \ +1) =$$
$$x^{13} + x^{11} + x^9 + x^8 + x^6 + x^5 + x^4 + x^3 + 1 \equiv$$
$$(x^7 + x^4 + x^2 + x) \bmod (x^8 + x^7 + x^5 + x^3 + 1)$$

Con bytes implica aplicar op. distributiva, y luego sucesivos desplazamientos a izquierda de forma que si el bit + significativo antes de desplazar es un 1, entonces tras desplazar se hace una XOR con el byte A9=10101001.

Ej: 01010111x10000011=01010111x00000001+01010111x00000010+01010111x10000000=

01010111+10101110+01010111x10000000=01010111+10101110+01101111=**10010110=96**

0	1	2	3	4	5	6	7
01010111	10101110	01011100+ 10101001= 11110101	11101010+ 10101001= 01000011	10000110	00001100+ 10101001= 10100101	01001010+ 10101001= 11100011	11000110+ 10101001= 01101111

Multiplicaciones en SNOW 3G

If el bit más más significativo de la entrada V = 1, then

$$MULx(V, c) = (V <<_8 1) \oplus c,$$

else

$$MULx(V, c) = V <<_8 1.$$

If i = 0, then

$$MULxPOW(V, i, c) = V,$$

else

$$MULxPOW(V, i, c) = MULx(MULxPOW(V, i - 1, c), c).$$

Multiplicaciones en SNOW 3G

$MUL_{\alpha}(c) =$

$(MULxPOW(c, 23, 0xA9) \,||\, MULxPOW(c, 245, 0xA9) \,||\, MULxPOW(c, 48, 0xA9) \,||\, MULxPOW(c, 239, 0xA9))$.

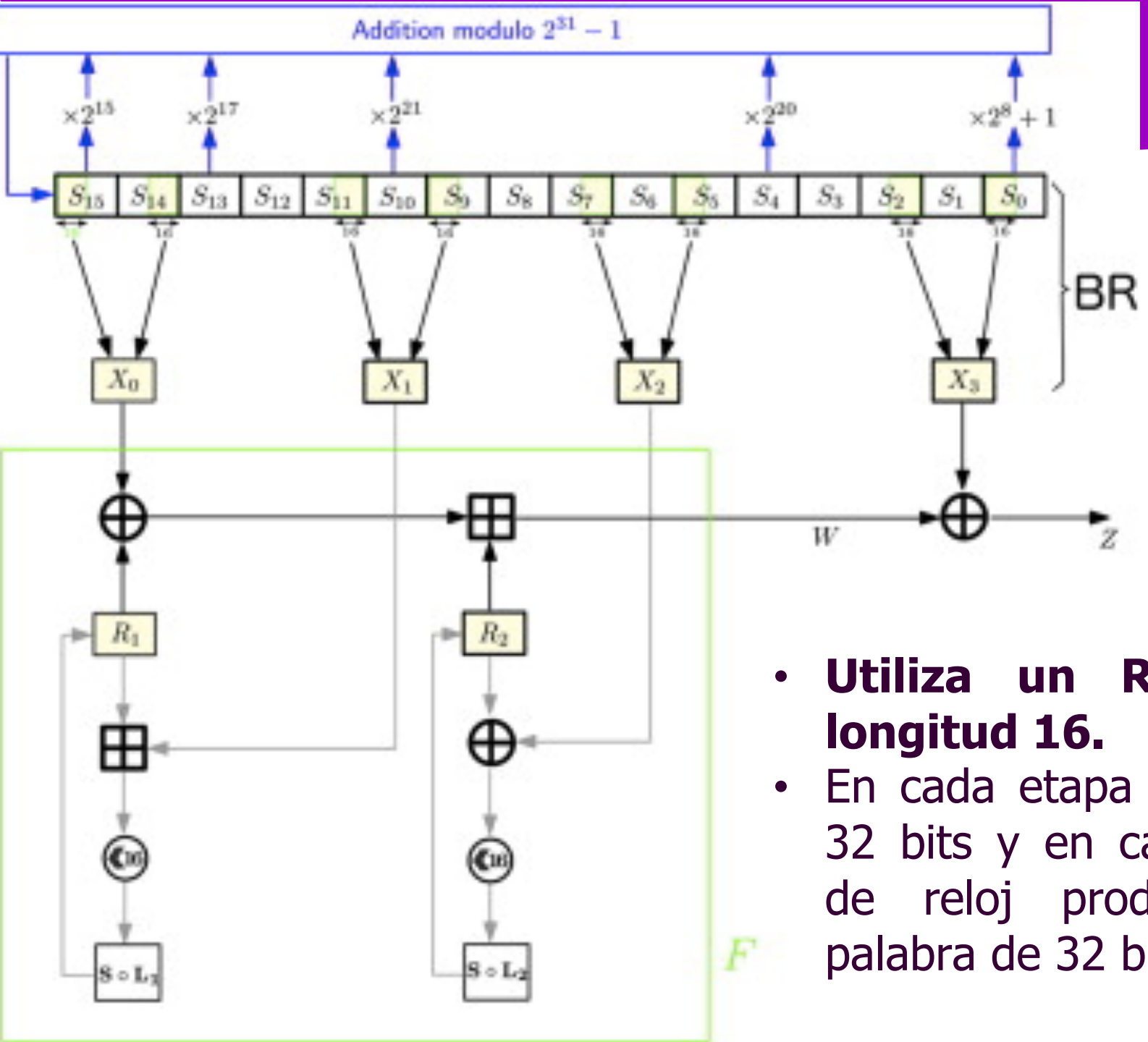
$DIV_{\alpha}(c) =$

$(MULxPOW(c, 16, 0xA9) \,||\, MULxPOW(c, 39, 0xA9) \,||\, MULxPOW(c, 6, 0xA9) \,||\, MULxPOW(c, 64, 0xA9))$.

Realimentación del LFSR =

$(s_{0,1} \,||\, s_{0,2} \,||\, s_{0,3} \,||\, 0x00) \oplus MUL_{\alpha}(s_{0,0}) \oplus s_2 \oplus (0x00 \,||\, s_{11,0} \,||\, s_{11,1} \,||\, s_{11,2}) \oplus DIV_{\alpha}(s_{11,3})$.

ZUC



- **Utiliza un RDRL de longitud 16.**
- En cada etapa almacena 32 bits y en cada golpe de reloj produce una palabra de 32 bits

Inseguridad de 4G vs 5G

- En **2018** usando una herramienta llamada LTEInspector se atacó en 4G:
  - Conexión: cuando se asocia el dispositivo de un usuario con la red.
  - Desconexión: cuando se apaga el dispositivo y se desconecta de la red.
  - Búsqueda: cuando se realiza una llamada y se busca al dispositivo en la red.
- El ataque más peligroso a 4G es de **suplantación o spoofing** porque un atacante puede conectarse sin tener credenciales legítimas.
- Otro ataque a 4G permite **bloquear** a un dispositivo el acceso a la red, **rastrear** a un usuario por las antenas que usa, o enviar alertas falsas.
- **5G se empezó a implantar en 2020.**
- La mayor diferencia entre 5G y 4G es la velocidad, pero en seguridad:
  - soporta claves de 256 bits para versiones futuras,
  - comprueba integridad detectando cambios no autorizados de los datos del usuario,
  - hay cambios en la autenticación AKA,
  - protege la confidencialidad de los mensajes iniciales entre dispositivo y red,
  - permite reautenticación al moverse entre diferentes redes,
  - se puede usar la clave pública de la red para cifrar parte del identificador de abonado,
  - permite verificar la presencia, lo que resulta útil para roaming y prevención de fraudes,
  - se puede utilizar autenticación mutua basada en certificados,
  - protege el perímetro de la red doméstica en las interconexiones con otras redes.